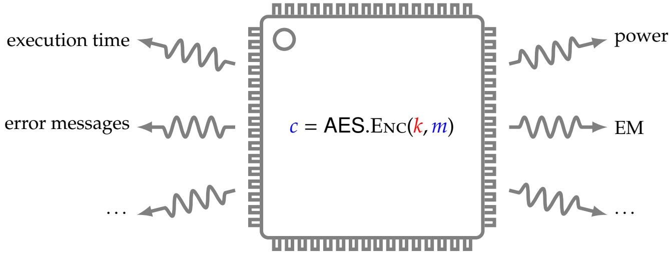
FENL: an ISE to mitigate analogue micro-architectural leakage

Si Gao, Ben Marshall, **Daniel Page**, and Thinh Pham
Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol BS8 1UB, United Kingdom.
`{si.gao,ben.marshall,daniel.page,th.pham}@bristol.ac.uk`

25/08/20

Background

- Context: **information leakage** ($\Rightarrow$ **side-channel attack**).

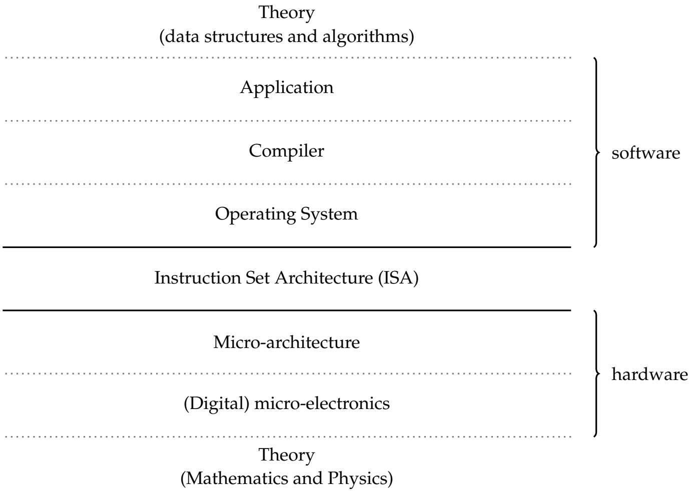


which can be (somewhat imprecisely) characterised as

•	scalar	vs.	vector
•	discrete	vs.	analogue
•	local	vs.	remote
•	stand-alone	vs.	supported by additional equipment
		⋮	

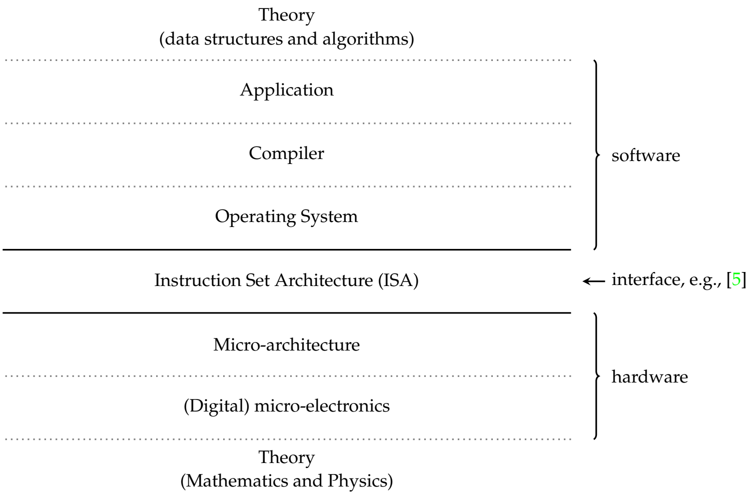
Background

- Context: **information leakage** (⇒ **side-channel attack**).


Theory
(data structures and algorithms)
Application
Compiler
Operating System
software
Instruction Set Architecture (ISA)
Micro-architecture
(Digital) micro-electronics
hardware
Theory
(Mathematics and Physics)

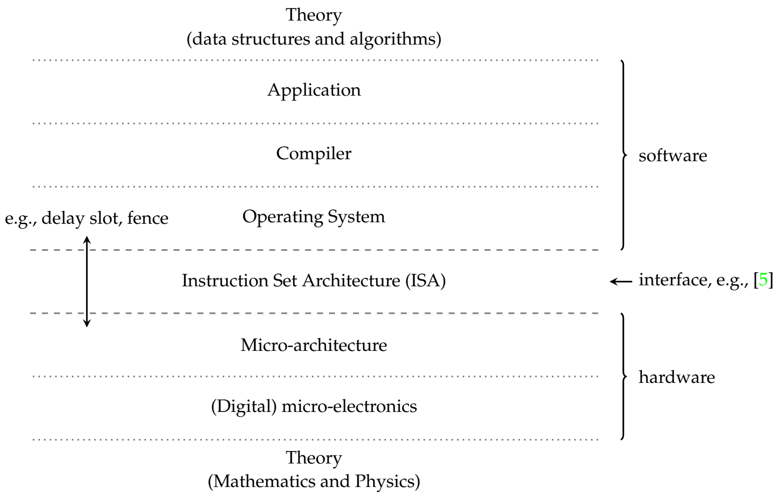
Background

- Context: **information leakage** (⇒ **side-channel attack**).


Theory
(data structures and algorithms)
Application
Compiler
Operating System
software
Instruction Set Architecture (ISA)
interface, e.g., [5]
Micro-architecture
(Digital) micro-electronics
hardware
Theory
(Mathematics and Physics)

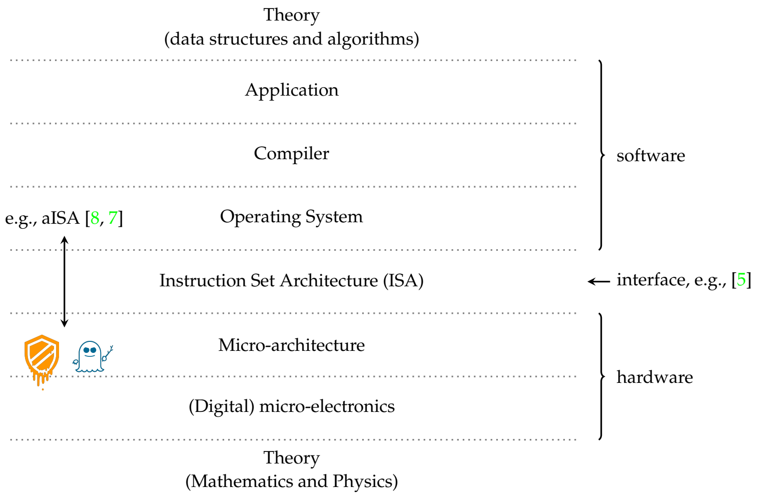
Background

- Context: **information leakage** (⇒ **side-channel attack**).



Background

- Context: **information leakage** ($\Rightarrow$ **side-channel attack**).


Theory
(data structures and algorithms)
Application
Compiler
Operating System
e.g., aISA [8, 7]
software
Instruction Set Architecture (ISA)
interface, e.g., [5]
Micro-architecture
(Digital) micro-electronics
hardware
Theory
(Mathematics and Physics)

Background

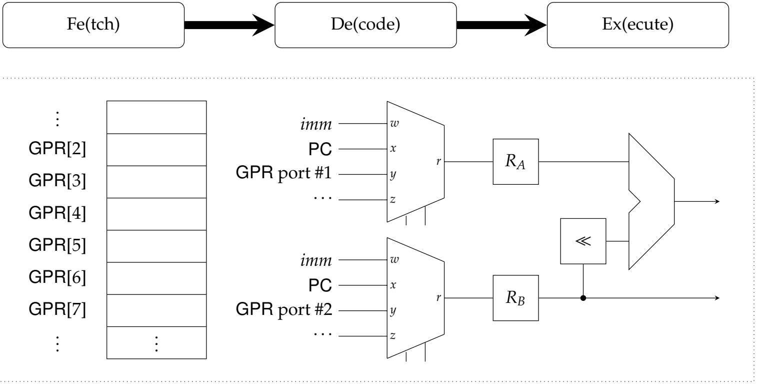
- Problem: ***analogue* information leakage**.
- (A) solution: **masking** (in theory).
  1. alter *representation*, e.g., via a 1-st order Boolean (vs. arithmetic) mask:
     - select random m,
     - represent variable x as
       $$\hat{x} = (\hat{x}^0 = x \oplus m, \hat{x}^1 = m).$$
  2. alter *computation*, e.g., instead of $x \wedge y$ we could use [2, Table 1]
     $$\mathsf{SecAnd}(\hat{x}, \hat{y}) = \hat{r} = (\hat{r}^0, \hat{r}^1)$$
     where
     $$\begin{array}{rcl} \hat{r}^0 & = & (\hat{x}^0 \wedge \hat{y}^1) \oplus (\hat{x}^0 \vee \neg \hat{y}^0) \\ \hat{r}^1 & = & (\hat{x}^1 \wedge \hat{y}^1) \oplus (\hat{x}^1 \vee \neg \hat{y}^0) \end{array}$$
     avoiding interaction between, e.g., $\hat{x}^0$ and $\hat{x}^1$.

Background

- Problem: ***analogue* information leakage**.
- (A) solution: **masking** (in practice, e.g., per [3, 6] on an ARM Cortex-M3 [4]).
  - ARMv7-M (including Thumb-2) ISA, and
  - micro-architecture based on 3-stage pipeline [4, Figure 1.2].

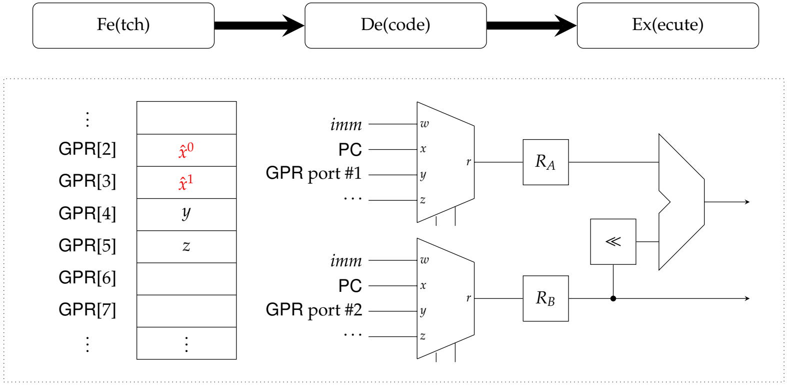
Background

- Problem: ***analogue* information leakage**.
- (A) solution: **masking** (in practice, e.g., per [3, 6] on an ARM Cortex-M3 [4]).



Background

- Problem: ***analogue* information leakage**.
- (A) solution: **masking** (in practice, e.g., per [3, 6] on an ARM Cortex-M3 [4]).

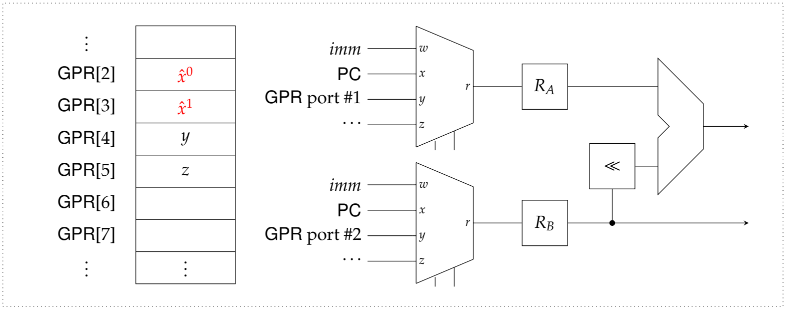


Background

- Problem: ***analogue* information leakage**.
- (A) solution: **masking** (in practice, e.g., per [3, 6] on an ARM Cortex-M3 [4]).

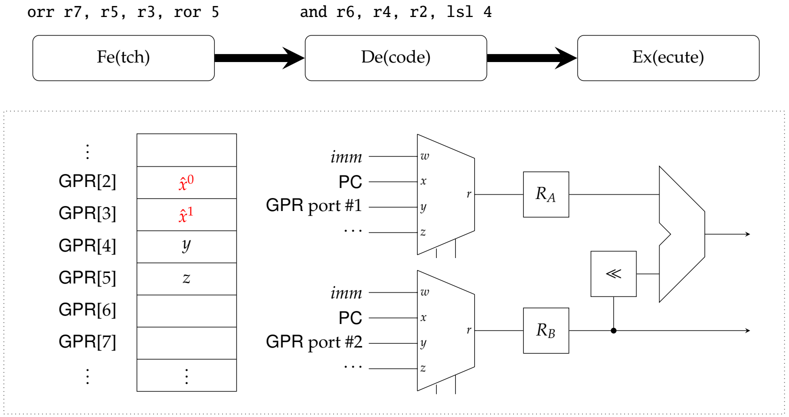
```
and r6, r4, r2, lsl 4
```

$$\begin{tikzcd}\boxed{\text{Fe(tch)}} \arrow[r] & \boxed{\text{De(code)}} \arrow[r] & \boxed{\text{Ex(ecute)}}\end{tikzcd}$$



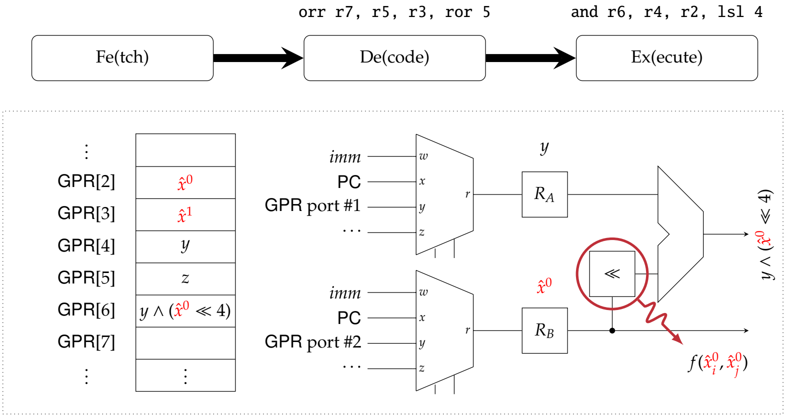
Background

- Problem: ***analogue* information leakage**.
- (A) solution: **masking** (in practice, e.g., per [3, 6] on an ARM Cortex-M3 [4]).



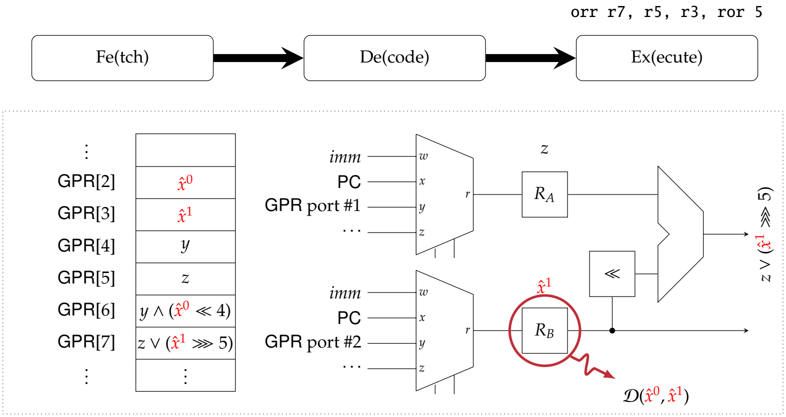
Background

- Problem: ***analogue* information leakage**.
- (A) solution: **masking** (in practice, e.g., per [3, 6] on an ARM Cortex-M3 [4]).



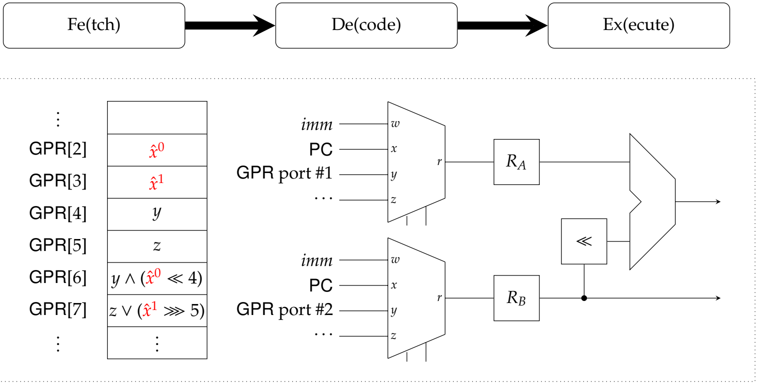
Background

- Problem: ***analogue* information leakage**.
- (A) solution: **masking** (in practice, e.g., per [3, 6] on an ARM Cortex-M3 [4]).



Background

- Problem: ***analogue* information leakage**.
- (A) solution: **masking** (in practice, e.g., per [3, 6] on an ARM Cortex-M3 [4]).



FENL (at a high level)

- Concept:
  - consider a sequence of instructions

$$\ldots, \quad \mathcal{I}_0, \quad \mathcal{I}_1, \quad \ldots, \quad \mathcal{I}_{i-1}, \quad \mathcal{I}_i, \quad \mathcal{I}_{i+1}, \quad \ldots, \quad \mathcal{I}_{n-2}, \quad \mathcal{I}_{n-1}, \quad \ldots$$

FENL (at a high level)

- Concept:
  - consider a sequence of instructions

$$\underbrace{\ldots, \quad \mathcal{I}_0, \quad \mathcal{I}_1, \quad \ldots, \quad \mathcal{I}_{i-1},}_{\text{before fence, } \mathcal{I}_j \text{ for } j < i} \quad \underbrace{\mathcal{I}_i,}_{\text{fence, } \mathcal{I}_i} \quad \underbrace{\mathcal{I}_{i+1}, \quad \ldots, \quad \mathcal{I}_{n-2}, \quad \mathcal{I}_{n-1}, \quad \ldots}_{\text{after fence, } \mathcal{I}_k \text{ for } k > i}$$

    st. a **fence** (or **barrier**) instruction $\mathcal{I}_i$ controls interaction between a given $\mathcal{I}_j$ and $\mathcal{I}_k$.

FENL (at a high level)

- Concept:
  - consider a sequence of instructions

$$\underbrace{\ldots, \quad \mathcal{I}_0, \quad \mathcal{I}_1, \quad \ldots, \quad \mathcal{I}_{i-1},}_{\text{before fence, } \mathcal{I}_j \text{ for } j < i} \quad \underbrace{\mathcal{I}_i,}_{\text{fence, } \mathcal{I}_i} \quad \underbrace{\mathcal{I}_{i+1}, \quad \ldots, \quad \mathcal{I}_{n-2}, \quad \mathcal{I}_{n-1}, \quad \ldots}_{\text{after fence, } \mathcal{I}_k \text{ for } k > i}$$

    st. a **fence** (or **barrier**) instruction $\mathcal{I}_i$ controls interaction between a given $\mathcal{I}_j$ and $\mathcal{I}_k$,
  - e.g., memory access
    - x86: `mfence` [10, Page 4-22], `sfence` [10, Page 4-597], and `lfence` [10, Page 3-541]
    - ARM: `dmb` [1, Section A6.7.21]
    - SPARC: `membar` [14, Section 8.4.3]
    - MIPS: `sync` [12, Pages 407–411]
    - RISC-V: `fence` [13, Section 2.7]

FENL (at a high level)

- Concept:
  - consider a sequence of instructions

$$\underbrace{\ldots, \quad \mathcal{I}_0, \quad \mathcal{I}_1, \quad \ldots, \quad \mathcal{I}_{i-1},}_{\text{before fence, } \mathcal{I}_j \text{ for } j < i} \quad \underbrace{\mathcal{I}_i,}_{\text{fence, } \mathcal{I}_i} \quad \underbrace{\mathcal{I}_{i+1}, \quad \ldots, \quad \mathcal{I}_{n-2}, \quad \mathcal{I}_{n-1}, \quad \ldots}_{\text{after fence, } \mathcal{I}_k \text{ for } k > i}$$

    st. a **fence** (or **barrier**) instruction $\mathcal{I}_i$ controls interaction between a given $\mathcal{I}_j$ and $\mathcal{I}_k$,

  - e.g., *analogue* information leakage $\rightsquigarrow$ **FENL** = **a FENce for Leakage**.

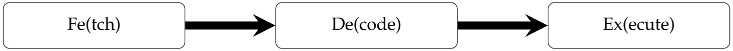
FENL (at a high level)

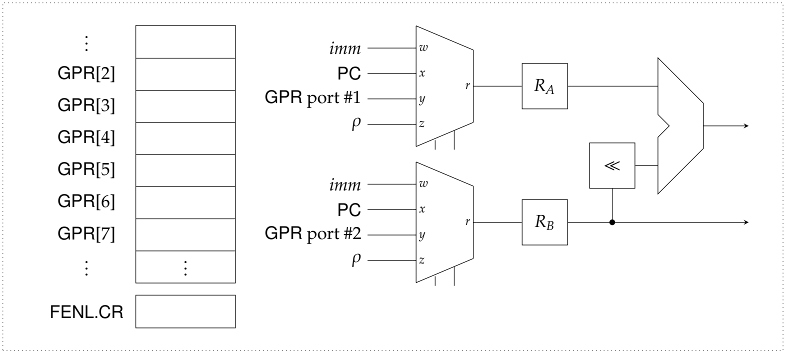
- Concept: given R, a set of micro-architectural resources, add
  1. a w-bit configuration register FENL.CR st.
  $$\mathsf{FENL.CR}_i \iff R_i,$$
  2. access instructions, e.g.,
  $$\begin{array}{lcl} \texttt{fenl.crwr rs} & \mapsto & \mathsf{FENL.CR} \leftarrow \mathsf{GPR}[\texttt{rs}] \\ \texttt{fenl.crrd rd} & \mapsto & \mathsf{GPR}[\texttt{rd}] \leftarrow \mathsf{FENL.CR} \end{array}$$
  and
  3. a fence instruction `fenl.fence` st.
     - when fence reaches execution stage j,
     - each i-th resource that exists or is used in stage j is flushed iff. $\mathsf{FENL.CR}_i = 1$,

  per Property 1 of the aISA [7, Section 5].

FENL (at a high level)

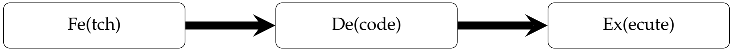
- Concept:

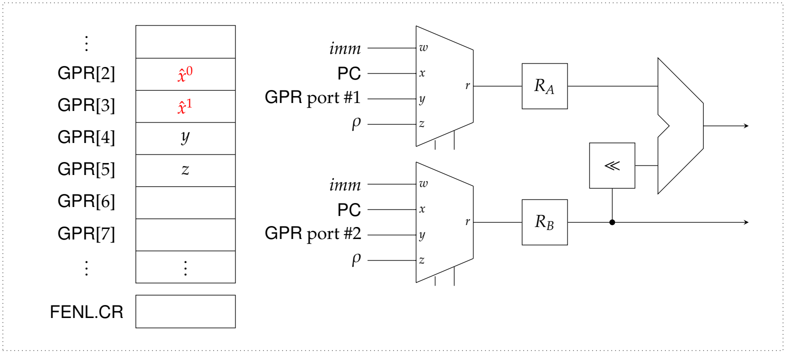




FENL (at a high level)

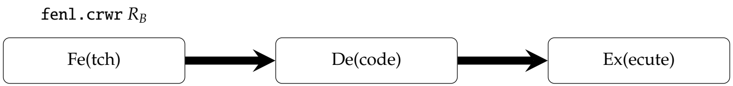
- Concept:

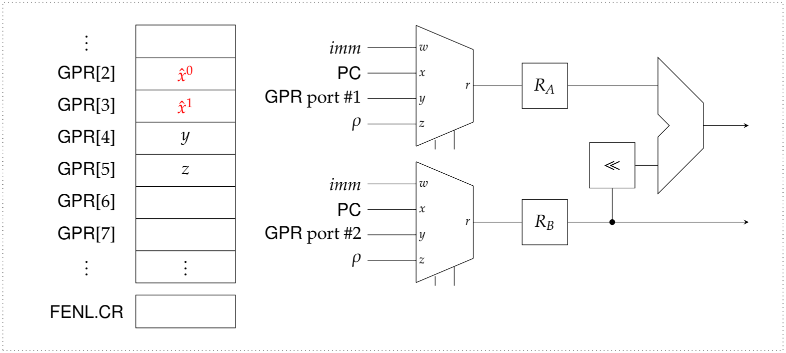




FENL (at a high level)

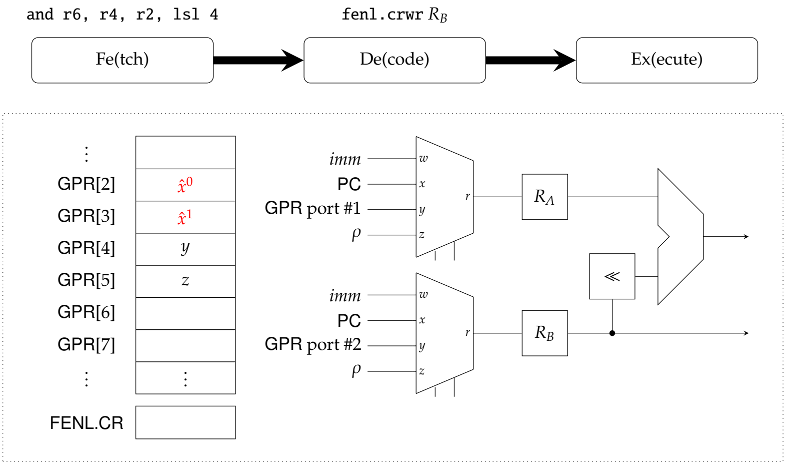
- Concept:





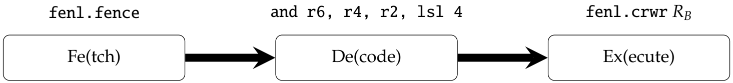
FENL (at a high level)

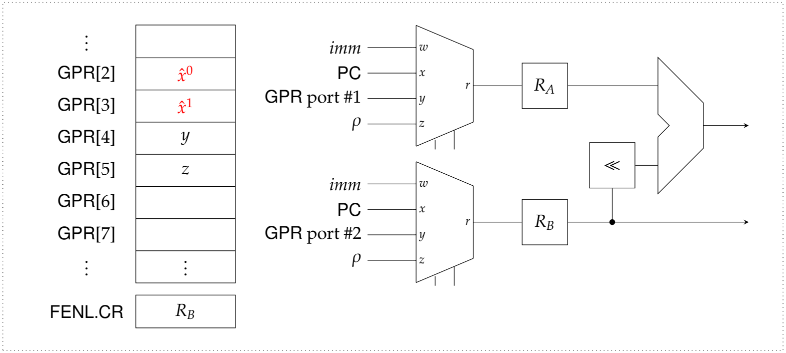
- Concept:



FENL (at a high level)

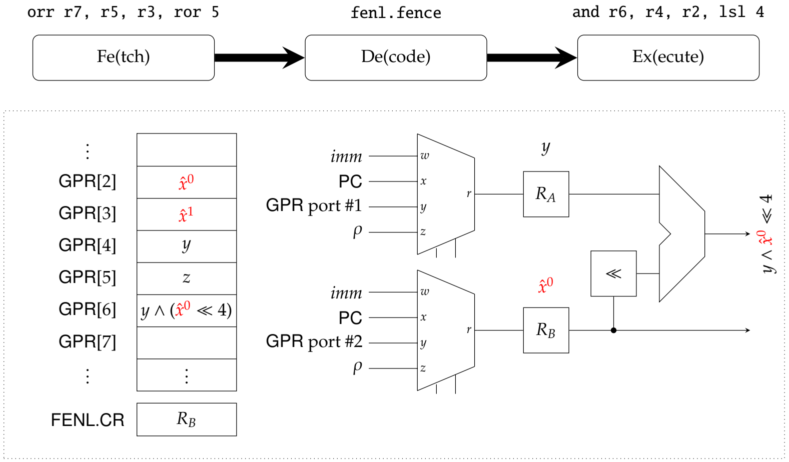
- Concept:





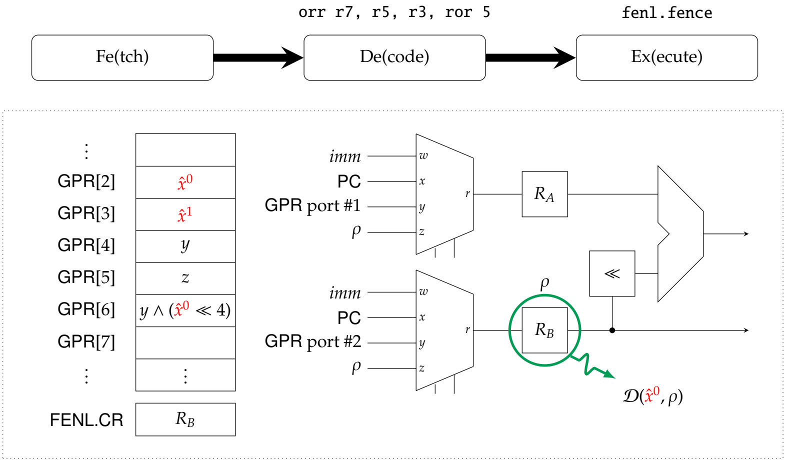
FENL (at a high level)

- Concept:



FENL (at a high level)

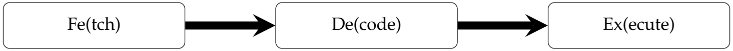
- Concept:

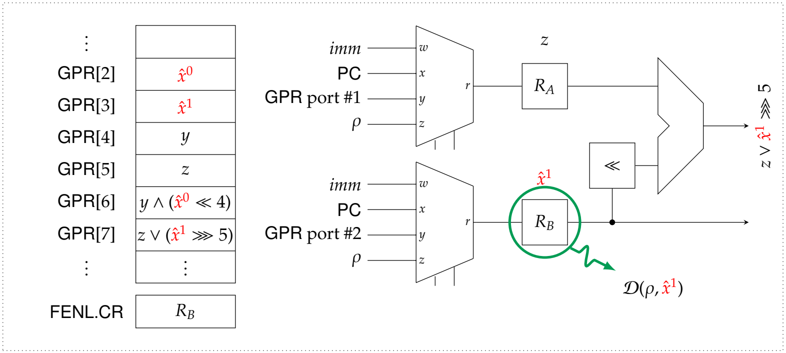


FENL (at a high level)

- Concept:

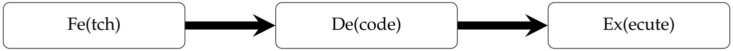
`orr r7, r5, r3, ror 5`

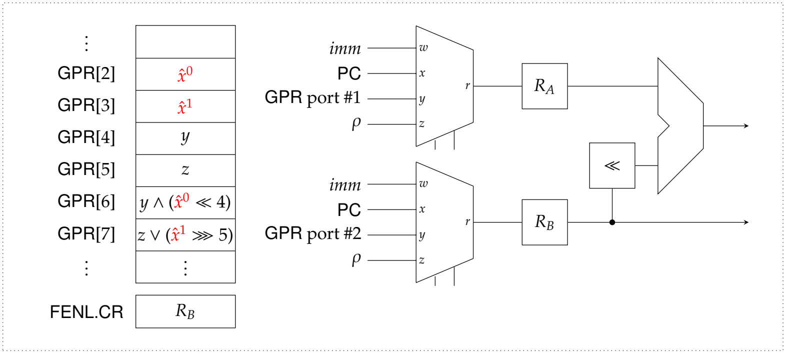




FENL (at a high level)

- Concept:





FENL (at a low level)

Quote

– *Hackers* (https://www.imdb.com/title/tt0113243)

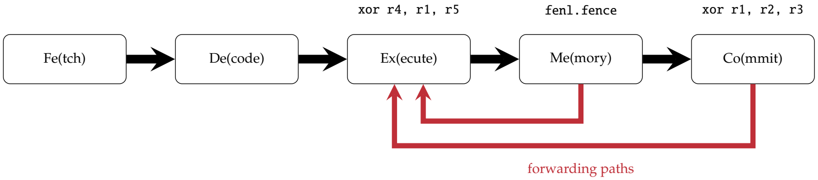
https://imgur.com/t/hackers/YZMw45k

FENL (at a low level)

- Complications:
  1. flush semantics: we *could* flush using a

     $$\rho \in \{ \text{zeroise}, \text{randomise with PRNG}, \text{randomise with TRNG}, \ldots \}$$

  2. forwarding paths



     require careful management.
  3. ...

FENL (at a low level)

- Implementation(s): we investigated *two* RISC-V cores, namely
  1. PicoRV32:
     - RV32IMC + XCrypto + FENL
     - non-pipelined, multi-cycle micro-architecture
     - no caches
  2. SCARV:
     - RV32IMC + XCrypto + FENL
     - 5-stage, in-order pipelined micro-architecture
     - no caches

  embedded in a uniform SoC (e.g., including memory, UART).

https://github.com/cliffordwolf/picorv32

https://github.com/scarv/scarv

FENL (at a low level)

- Implementation(s): we synthesised the cores for a Xilinx Kintex-7 FPGA
  1. PicoRV32 (@ 25 MHz):

	Baseline	Baseline + FENL with zeroisation	Baseline + FENL with randomisation
Slice LUTs	1977	2002 (+1.3%)	2005 (+1.4%)
Slice FFs	1098	1100 (+0.1%)	1139 (+3.7%)

  2. SCARV (@ 25 MHz):

	Baseline	Baseline + FENL with zeroisation	Baseline + FENL with randomisation
Slice LUTs	5952	6014 (+1.0%)	6173 (+3.1%)
Slice FFs	2147	2163 (+0.7%)	2193 (+2.1%)

  on the SASEBO-GIII [9] side-channel analysis platform.

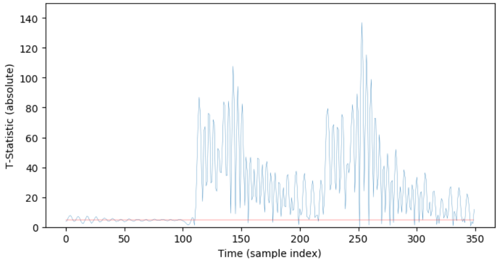
https://github.com/cliffordwolf/picorv32

https://github.com/scarv/scarv

FENL (at a low level)

- Example #1 (2-share ISW [11] multiplication):

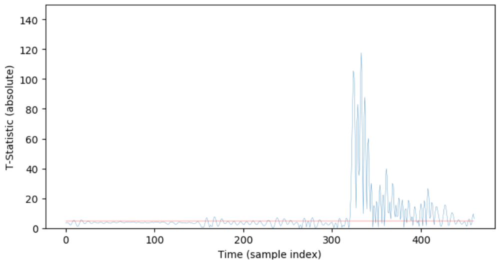
```
isw: lw      t0, 0(a0)
     lw      t2, 0(a1)
     lw      t1, 4(a0)
     lw      t3, 4(a1)
     and     t4, t0, t2

     and     t5, t1, t3

     xc.rngsamp t6
     xor     t6, t4, t6
     sw      t6, 0(a2)
     and     t0, t0, t3

     and     t1, t1, t2

     xor     t0, t0, t6
     xor     t0, t0, t1
     xor     t5, t5, t0
     sw      t5, 4(a2)
     ret
```



FENL (at a low level)

- Example #1 (2-share ISW [11] multiplication):

```
isw: lw      t0, 0(a0)
     lw      t2, 0(a1)
     lw      t1, 4(a0)
     lw      t3, 4(a1)
     and     t4, t0, t2
     fenl.fence          // fenl.cr = 1FFF
     and     t5, t1, t3
     fenl.fence          // fenl.cr = 1FFF
     xc.rngsamp t6
     xor     t6, t4, t6
     sw      t6, 0(a2)
     and     t0, t0, t3

     and     t1, t1, t2

     xor     t0, t0, t6
     xor     t0, t0, t1
     xor     t5, t5, t0
     sw      t5, 4(a2)
     ret
```



noting that

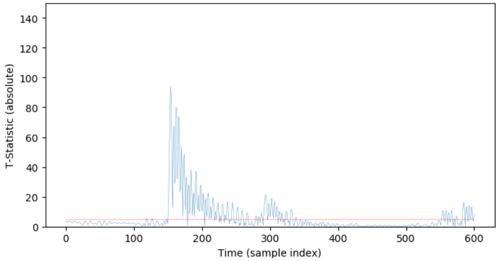
$$\text{FENL.CR} = 1FFF_{(16)}$$

$$\mapsto \left\{ \begin{array}{l} \texttt{s2\_opr\_a}, \texttt{s2\_opr\_b}, \texttt{s2\_opr\_c}, \texttt{s3\_opr\_a}, \texttt{s3\_opr\_b}, \texttt{s4\_opr\_a}, \texttt{s4\_opr\_b}, \\ \texttt{fu\_mult}, \texttt{fu\_aessub}, \texttt{fu\_aesmix}, \\ \texttt{uncore\_0}, \texttt{uncore\_1}, \texttt{uncore\_2} \end{array} \right\}$$

FENL (at a low level)

- Example #1 (2-share ISW [11] multiplication):

```
isw: lw      t0, 0(a0)
     lw      t2, 0(a1)
     lw      t1, 4(a0)
     lw      t3, 4(a1)
     and     t4, t0, t2
     fenl.fence          // fenl.cr = 1C14
     and     t5, t1, t3
     fenl.fence          // fenl.cr = 1C14
     xc.rngsamp t6
     xor     t6, t4, t6
     sw      t6, 0(a2)
     and     t0, t0, t3
     fenl.fence          // fenl.cr = 1C14
     and     t1, t1, t2
     fenl.fence          // fenl.cr = 1C14
     xor     t0, t0, t6
     xor     t0, t0, t1
     xor     t5, t5, t0
     sw      t5, 4(a2)
     ret
```



noting that

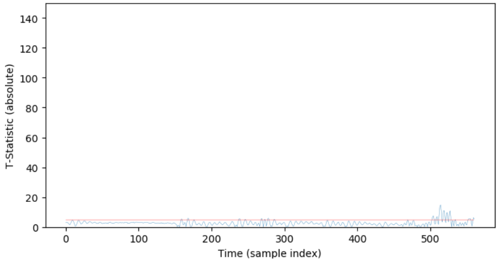
$$\text{FENL.CR} = 1C14_{(16)}$$

$$\mapsto \left\{ \begin{array}{l} \texttt{s2\_opr\_c}, \texttt{s3\_opr\_b}, \texttt{s4\_opr\_b}, \\ \texttt{uncore\_0}, \texttt{uncore\_1}, \texttt{uncore\_2} \end{array} \right\}$$

FENL (at a low level)

- Example #1 (2-share ISW [11] multiplication):

```
isw: lw      t0, 0(a0)
     lw      t2, 0(a1)
     lw      t1, 4(a0)
     lw      t3, 4(a1)
     and     t4, t0, t2
     fenl.fence          // fenl.cr = 1FFF
     and     t5, t1, t3
     fenl.fence          // fenl.cr = 1FFF
     xc.rngsamp t6
     xor     t6, t4, t6
     sw      t6, 0(a2)
     and     t0, t0, t3
     fenl.fence          // fenl.cr = 1FFF
     and     t1, t1, t2
     fenl.fence          // fenl.cr = 1FFF
     xor     t0, t0, t6
     xor     t0, t0, t1
     xor     t5, t5, t0
     sw      t5, 4(a2)
     ret
```



noting that

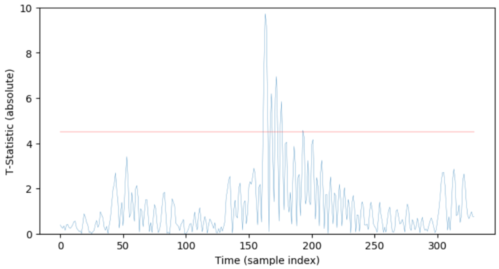
$$\text{FENL.CR} = 1FFF_{(16)}$$

$$\mapsto \left\{ \begin{array}{l} \texttt{s2\_opr\_a}, \texttt{s2\_opr\_b}, \texttt{s2\_opr\_c}, \texttt{s3\_opr\_a}, \texttt{s3\_opr\_b}, \texttt{s4\_opr\_a}, \texttt{s4\_opr\_b}, \\ \texttt{fu\_mult}, \texttt{fu\_aessub}, \texttt{fu\_aesmix}, \\ \texttt{uncore\_0}, \texttt{uncore\_1}, \texttt{uncore\_2} \end{array} \right\}$$

FENL (at a low level)

- Example #2 (sequential load $\rightsquigarrow$ buffer in AXI-based IP core):

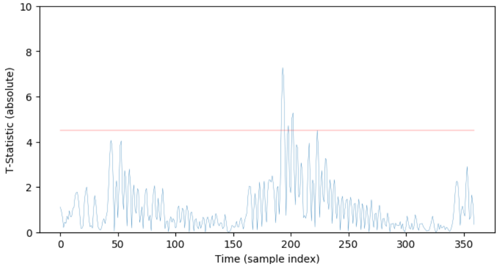
```
axi: lw      a3, 0(a0)

     lw      a4, 4(a0)
     ret
```



FENL (at a low level)

- Example #2 (sequential load $\rightsquigarrow$ buffer in AXI-based IP core):

```
axi: lw      a3, 0(a0)
     fenl.fence          // fenl.cr = 03FF
     lw      a4, 4(a0)
     ret
```



noting that

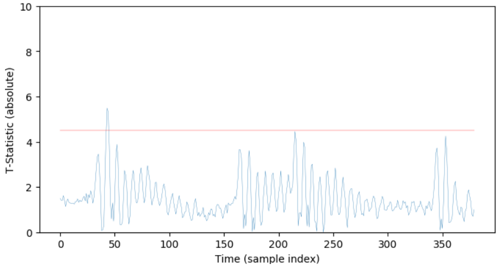
$$\text{FENL.CR} = 03FF_{(16)}$$

$$\mapsto \left\{ \begin{array}{l} \texttt{s2\_opr\_a}, \texttt{s2\_opr\_b}, \texttt{s2\_opr\_c}, \texttt{s3\_opr\_a}, \texttt{s3\_opr\_b}, \texttt{s4\_opr\_a}, \texttt{s4\_opr\_b}, \\ \texttt{fu\_mult}, \texttt{fu\_aessub}, \texttt{fu\_aesmix} \end{array} \right\}$$

FENL (at a low level)

- Example #2 (sequential load $\rightsquigarrow$ buffer in AXI-based IP core):

```
axi: lw      a3, 0(a0)
     fenl.fence        // fenl.cr = 1C00
     lw      a4, 4(a0)
     ret
```



noting that

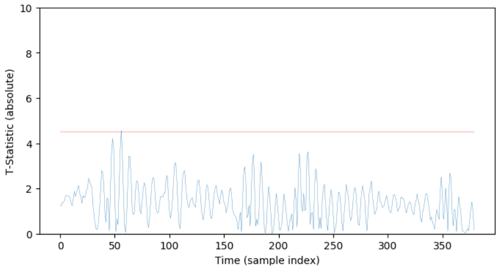
$$\text{FENL.CR} = 1C00_{(16)}$$

$$\mapsto \left\{ \texttt{uncore\_0}, \texttt{uncore\_1}, \texttt{uncore\_2} \right\}$$

FENL (at a low level)

- Example #2 (sequential load $\rightsquigarrow$ buffer in AXI-based IP core):

```
axi: lw       a3, 0(a0)
     fenl.fence          // fenl.cr = 1FFF
     lw       a4, 4(a0)
     ret
```



noting that

$$\begin{array}{lcl} \textsf{FENL.CR} & = & 1FFF_{(16)} \\ & \mapsto & \left\{ \begin{array}{l} \texttt{s2\_opr\_a}, \texttt{s2\_opr\_b}, \texttt{s2\_opr\_c}, \texttt{s3\_opr\_a}, \texttt{s3\_opr\_b}, \texttt{s4\_opr\_a}, \texttt{s4\_opr\_b}, \\ \texttt{fu\_mult}, \texttt{fu\_aessub}, \texttt{fu\_aesmix}, \\ \texttt{uncore\_0}, \texttt{uncore\_1}, \texttt{uncore\_2} \end{array} \right\} \end{array}$$

Conclusion

- FENL is a simple concept, an implementation of which can act as
  1. a mechanism for *localisation* of leakage,
  2. a mechanism for *reduction* of leakage,
  3. an practical “anchor” for theoretical reasoning about leakage,

  with relatively low design- and run-time overhead.

Conclusion

- FENL is a simple concept, an implementation of which can act as
  1. a mechanism for *localisation* of leakage,
  2. a mechanism for *reduction* of leakage,
  3. an practical “anchor” for theoretical reasoning about leakage,

  with relatively low design- and run-time overhead, *but* ...
- ... so far it’s a first step vs. a complete solution:
  - more complex micro-architectural designs (e.g., out-of-order),
  - use as an anchor in proofs, or exploration of alternate anchors,
  - (semi-)automation and verification, of implementation and/or use,
  - relationship with subsequent related work, e.g., [15],
  - ...

Conclusion

Questions?

Extra

PicoRV32 core

i	R_i	$\sigma(R_i)$	Description
0	`mem_wdata`	Operand Read	Memory write data register
1	`reg_op1`	Operand Read	Register read data 1 (RS1)
2	`reg_op2`	Operand Read	Register read data 2 (RS2)
3	`reg_out`	Operand Read	Register write data
4	`alu_out_q`	Operand Read	ALU output register
5	`uncore_0`	Operand Read	Un-core resource 0
6	`uncore_1`	Operand Read	Un-core resource 1

SCARV core

i	R_i	$\sigma(R_i)$	Description
0	`s2_opr_a`	Decode	Decode $\Rightarrow$ Execute pipeline register A
1	`s2_opr_b`	Decode	Decode $\Rightarrow$ Execute pipeline register B
2	`s2_opr_c`	Decode	Decode $\Rightarrow$ Execute pipeline register C
3	`s3_opr_a`	Execute	Execute $\Rightarrow$ Write memory result pipeline register A
4	`s3_opr_b`	Execute	Execute $\Rightarrow$ Write memory result pipeline register B
5	`fu_mult`	Execute	Multiply-accumulate intermediate state registers
6	`fu_aessub`	Execute	AES `SubBytes` intermediate state registers
7	`fu_aesmix`	Execute	AES `MixColumns` intermediate state registers
8	`s4_opr_a`	Memory	Memory $\Rightarrow$ Write-back result pipeline register A
9	`s4_opr_b`	Memory	Memory $\Rightarrow$ Write-back result pipeline register B
10	`uncore_0`	Memory	Un-core resource 0
11	`uncore_1`	Memory	Un-core resource 1
12	`uncore_2`	Memory	Un-core resource 2

References

[1] *ARMv6-M Architecture Reference Manual*. DDI0419E (issue E). ARM. 2018. URL: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0419c/index.html (see pp. 16–19).

[2] A. Biryukov et al. "Optimal First-Order Boolean Masking for Embedded IoT Devices". In: *Smart Card Research and Advanced Applications (CARDIS)*. LNCS 10728. Springer-Verlag, 2017, pp. 22–41 (see p. 7).

[3] Y. Le Corre, J. Großschädl, and D. Dinu. "Micro-architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors". In: *Constructive Side-Channel Analysis and Secure Design (COSADE)*. LNCS 10815. Springer-Verlag, 2018, pp. 82–98 (see pp. 8–15).

[4] *Cortex-M3 Technical Reference Manual*. Tech. rep. DDI-0337E. ARM Ltd., 2006. URL: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/index.html (see pp. 8–15).

[5] C. Dunham and J. Beard. "This Architecture Tastes Like Microarchitecture". In: *Workshop on Pioneering Processor Paradigms (WP3)*. 2018. URL: http://www.jonathanbeard.io/pdf/db18a.pdf (see pp. 4–6).

[6] S. Gao et al. "Share slicing: friend or foe?" In: *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)* 2020.1 (2019), pp. 152–174 (see pp. 8–15).

[7] Q. Ge, Y. Yarom, and G. Heiser. "No security without time protection: we need a new hardware-software contract". In: *Asia-Pacific Workshop on Systems (APSys)*. 2018 (see pp. 6, 20).

[8] G. Heiser. "For Safety's Sake: We Need a New Hardware-Software Contract!" In: *IEEE Design & Test* 35.2 (2018), pp. 27–30 (see p. 6).

[9] Y. Hori et al. "SASEBO-GIII: A hardware security evaluation board equipped with a 28-nm FPGA". In: *IEEE Global Conference on Consumer Electronics*. 2012, pp. 657–660 (see p. 33).

[10] *Intel 64 and IA-32 architectures – Software Developer's Manual (Volume 2: Instruction Set Reference A-Z)*. Tech. rep. 325383-067US. Intel Corp., 2018. URL: http://software.intel.com/en-us/articles/intel-sdm (see pp. 16–19).

[11] Y. Ishai, A. Sahai, and D. Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: *Advances in Cryptology (CRYPTO)*. LNCS 2729. Springer-Verlag, 2003, pp. 463–481 (see pp. 34–37).

[12] *MIPS Architecture for Programmers Volume II-A: The MIPS32 Instruction Set Manual*. Tech. rep. MD00086 (rev. 6.06). MIPS, 2016. URL: https://www.mips.com/products/architectures/mips32-2 (see pp. 16–19).

References

[13] *.The RISC-V Instruction Set Manual*. Tech. rep. Volume I: User-Level ISA (Version 20190608-Base-Ratified). 2019. URL: `http://riscv.org/specifications/` (see pp. 16–19).

[14] D.L. Weaver and T. Germond, eds. *The SPARC Architecture Manual: Version 9*. Prentice-Hall, 2003. URL: `https://sparc.org/technical-documents` (see pp. 16–19).

[15] N. Wistoff et al. "Prevention of Microarchitectural Covert Channels on an Open-Source 64-bit RISC-V Core". In: *Computer Architecture Research with RISC-V (CARRV)*. 2020 (see pp. 42, 43).